

Generate

mount google drive



Close

< 1 of 1 > [Use code with caution](#)

prompt: mount google drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Mounted at /content/drive

!pip install openai

 Collecting openai
 Downloading openai-1.50.2-py3-none-any.whl.metadata (24 kB)
 Requirement already satisfied: anyio<5,>=3.5.0 in /usr/local/lib/python3.10/dist-packages (from openai) (3.7.1)
 Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-packages (from openai) (1.7.0)
 Collecting httpx<1,>=0.23.0 (from openai)
 Downloading httpx-0.27.2-py3-none-any.whl.metadata (7.1 kB)
 Collecting jiter<1,>=0.4.0 (from openai)
 Downloading jiter-0.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.6 kB)
 Requirement already satisfied: pydantic<3,>=1.9.0 in /usr/local/lib/python3.10/dist-packages (from openai) (2.9.2)
 Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-packages (from openai) (1.3.1)
 Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.5)
 Requirement already satisfied: typing-extensions<5,>=4.11 in /usr/local/lib/python3.10/dist-packages (from openai)
 Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai)
 Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio<5,>=3.5.0->openai)
 Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx<1,>=0.23.0->openai)
 Collecting httpcore==1.* (from httpx<1,>=0.23.0->openai)
 Downloading httpcore-1.0.5-py3-none-any.whl.metadata (20 kB)
 Collecting h11<0.15,>=0.13 (from httpcore==1.*->httpx<1,>=0.23.0->openai)
 Downloading h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
 Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,
 Requirement already satisfied: pydantic-core==2.23.4 in /usr/local/lib/python3.10/dist-packages (from pydantic<3,
 Downloading openai-1.50.2-py3-none-any.whl (382 kB)
 _____ 383.0/383.0 kB 30.4 MB/s eta 0:00:00
 Downloading httpx-0.27.2-py3-none-any.whl (76 kB)
 _____ 76.4/76.4 kB 7.9 MB/s eta 0:00:00
 Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
 _____ 77.9/77.9 kB 7.1 MB/s eta 0:00:00
 Downloading jiter-0.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (318 kB)
 _____ 318.9/318.9 kB 27.0 MB/s eta 0:00:00
 Downloading h11-0.14.0-py3-none-any.whl (58 kB)
 _____ 58.3/58.3 kB 5.6 MB/s eta 0:00:00
 Installing collected packages: jiter, h11, httpcore, httpx, openai
 Successfully installed h11-0.14.0 httpcore-1.0.5 httpx-0.27.2 jiter-0.5.0 openai-1.50.2

import copy

```
import torch
from torch import nn
from torchvision import datasets, transforms, utils
```

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from tqdm.notebook import tqdm
# from nullload import NullLoader
```

```
from torch.utils.data import Dataset, DataLoader
import pandas as pd
from openai import OpenAI
```

import pickle



This converts the private dataset in GDrive to a pickle file containing the embeddings mapped to race and gender

features vector (index 0)

Contains n-dimensional vector embedding representation of essay

label vector (index 1)

index 0 is race

- 0 for Hispanic
- 1 for Black
- 2 for American Indian or Alaskan Native
- 3 for Asian
- 4 for White

index 1 is gender

- 0 for Female
- 1 for Male

✓ Create a new pickle file dataset from private friends and family database

```
client = OpenAI(api_key="sk-proj-Ubt0AsJUaXfZ0b7f34CkVj_JiqpTYhma0sj5MwUD5yMVmFizaHnrgy4PEn29KHDCs7qxdXWyImT3B1bkFJUKX")
def private_dataset_to_pickle(input_file, output_file):
    db = pd.read_csv(input_file)
    new_data = []
    for row in db.iterrows():
        new_row = []

        races = ["Hispanic", "Black", "American Indian or Alaskan Native", "Asian", "White"]
        genders = ["Female", "Male"]
        embedding = client.embeddings.create(input=row[1][0], model="text-embedding-3-small", dimensions=256).data[0].embedding
        new_row.append(np.array(embedding))

        label = [races.index(row[1][1]), genders.index(row[1][2])]
        label = np.array(label)

        new_row.append(label)
        new_data.append(new_row)

    print("new_data", new_data)
    db = pd.DataFrame(new_data)
    db.to_pickle(output_file)
```

```
private_dataset_to_pickle('/content/drive/MyDrive/Diversify/essay_dataset.csv', '/content/drive/MyDrive/Diversify/essay_dataset.pkl')
```

 Show hidden output

✓ Load a pickle file dataset

```
class PickleEssaysDataset(Dataset):
    def __init__(self, pkl_file):
        with open(pkl_file, 'rb') as f:
            self.data = pickle.load(f)

    def __len__(self):
```

```

return len(self.data)


def __getitem__(self, idx):
    #indexed by column, row
    features = np.array(self.data[0][idx])
    label = np.array(self.data[1][idx])

    # Convert features to torch tensor
    features = torch.tensor(features, dtype=torch.double)
    label = torch.tensor(label, dtype=torch.int8)

    return features, label

private_dataset = PickleEssaysDataset(pk1_file='/content/drive/MyDrive/Diversify/essay_dataset.pkl')
private_dataset.__getitem__(0)

```

 (tensor([1.2201e-01, -9.2525e-02, -1.0475e-01, 8.2071e-02, 3.2064e-02, -8.5036e-02, -1.4685e-03, 4.7381e-02, -1.2212e-01, -5.4383e-03, 9.5243e-03, -2.6317e-02, -4.3662e-02, -6.3556e-02, 9.3409e-02, 3.1882e-02, 2.1116e-02, -1.3132e-02, -1.0252e-02, 7.3707e-04, 5.5390e-02, -5.3180e-02, 9.2005e-02, 6.4362e-03, 2.7409e-02, -9.3773e-02, 1.6741e-03, 6.9745e-02, 5.5130e-02, -5.2322e-02, 6.9043e-03, -2.4874e-02, -8.9249e-02, 1.8294e-02, 2.6395e-02, 9.8350e-02, 5.3778e-02, -6.8861e-02, 5.5026e-02, 5.6014e-02, 1.5174e-02, -4.1530e-02, 4.6497e-02, 3.1388e-02, -1.0288e-01, -4.9929e-02, -4.5743e-02, 3.0166e-02, 8.5972e-02, 3.1570e-02, 2.0206e-02, 5.7211e-02, 1.3210e-01, 1.3887e-01, -3.9943e-02, -2.2481e-02, -1.6409e-02, -4.2648e-03, -9.0861e-02, -5.6327e-02, 6.4180e-02, -2.5277e-02, 8.6596e-02, 4.7751e-03, 1.0584e-02, 5.3518e-02, -6.3764e-02, -1.5746e-02, 8.3931e-03, -2.7590e-04, 9.1173e-02, 1.2690e-01, -6.2802e-03, 1.5343e-02, -8.0355e-02, -1.0839e-01, -7.9523e-02, 2.5901e-02, -4.1010e-02, -7.3126e-02, -8.9912e-03, -1.9023e-02, 2.9776e-02, 1.8789e-02, -1.2087e-01, 4.0600e-03, -8.5920e-02, -1.3575e-01, 3.5003e-02, -1.4864e-01, -1.5369e-02, -6.3348e-02, 3.5939e-02, 6.0227e-02, 1.5041e-01, 1.8697e-02, -3.8130e-03, 4.1920e-02, 6.7041e-02, -3.2246e-02, 6.2360e-02, -6.0435e-02, 4.9929e-02, -7.4601e-04, 5.9194e-03, 8.7896e-02, 4.9695e-02, -1.9062e-02, -6.7821e-02, -7.2554e-03, -1.9961e-01, -4.7979e-02, -3.3936e-03, 4.8707e-02, -4.9357e-02, 3.1908e-02, -3.0582e-02, -5.0319e-02, 3.7811e-02, 1.8958e-02, -2.0271e-02, 6.8523e-03, 3.8773e-02, -2.7617e-02, 1.3419e-02, -8.1343e-02, 1.1026e-01, 2.2247e-02, -6.8809e-02, -6.2412e-02, 1.2762e-02, 1.7956e-02, -2.6512e-02, -2.4380e-02, -1.6045e-02, -7.2502e-02, -1.0397e-01, -3.8331e-02, 1.1618e-02, 3.7024e-03, -2.1545e-02, -1.9010e-02, 4.2544e-02, -5.9915e-02, 1.2784e-01, 9.2525e-02, 8.3320e-02, 5.9863e-02, 7.4114e-03, 2.8579e-02, 8.8989e-02, -2.9099e-02, -9.4138e-02, -1.2034e-02, 1.1026e-01, 3.5172e-03, 3.2272e-02, 5.3518e-02, -5.2140e-02, 2.0713e-02, 1.0059e-01, -1.2212e-01, -1.9358e-01, -5.2218e-02, -5.4610e-02, 7.6649e-03, -8.1915e-02, -3.4794e-02, -9.0081e-02, -6.6156e-02, -3.9059e-02, -4.1894e-02, -5.7003e-02, 4.2492e-02, 2.0461e-01, 1.1039e-02, -8.3007e-02, -7.5622e-02, -1.1085e-03, 5.8823e-02, 1.3744e-02, 1.8255e-02, -4.1894e-02, -7.0928e-03, 7.8990e-03, 5.9187e-02, -4.8499e-02, -2.7383e-02, 8.6752e-02, -1.7059e-02, 5.4454e-02, 8.7376e-02, 1.5018e-02, -6.8185e-02, 3.8513e-02, -6.9433e-02, 1.3543e-01, 2.1925e-03, 1.5478e-05, 2.3456e-02, 7.1097e-02, 4.6887e-02, 1.0662e-01, -9.3162e-03, 3.6771e-02, -2.3859e-02, -6.4388e-02, 3.9397e-02, -9.4502e-02, -6.8965e-02, 3.9761e-02, -1.0334e-01, 8.9249e-02, -1.0603e-02, -3.8513e-02, 1.5837e-02, 5.1438e-02, 9.7102e-02, 2.9568e-02, 4.2542e-04, -6.7613e-02, 3.7265e-02, 5.2972e-02, -9.8818e-03, 6.1371e-02, 3.4404e-02, -2.3372e-03, -7.0941e-02, -2.9012e-04, -4.7069e-02, 2.7903e-02, 5.4454e-02, -4.5691e-02, -6.1631e-02, -5.1204e-02, 6.3660e-02, -1.3406e-02, -4.5092e-02, 6.3712e-02, -7.6584e-03, -4.8707e-02, 2.9620e-02, 1.1213e-01, 4.4052e-02, -1.9530e-02, 2.1402e-02, -7.8327e-02, -6.0487e-02, 1.1189e-02, -2.9177e-02, 4.1816e-02, 3.7837e-02, -8.7685e-04, 7.3698e-02, -7.3022e-02, -5.8738e-03], dtype=torch.float64), tensor([0, 1], dtype=torch.int8))

> (Optional) Load larger ELLIPSE corpus dataset

[] ↪ 1 cell hidden

✓ Load Synthetic Database

```
synthetic_dataset = PickleEssaysDataset(pkl_file='/content/drive/MyDrive/Diversify/synthetic_dataset.pkl')
synthetic_dataset.__getitem__(0)
```

```
→ (tensor([ 1.1729e-01, -1.2117e-01,  1.9815e-02,  1.3948e-01,  4.3668e-02,
           -5.5081e-02,  2.5248e-03,  5.3051e-02, -4.9081e-02,  1.7070e-01,
           -2.8375e-02,  2.7112e-02, -7.3622e-02, -1.6759e-02,  1.0773e-01,
            8.8599e-02,  7.1502e-03,  7.5336e-02,  4.7638e-02,  5.5577e-02,
            3.8074e-02, -1.0725e-02,  9.3183e-04,  2.9351e-03, -3.2345e-02,
           -7.8404e-02,  1.4639e-02,  2.8804e-02,  6.8457e-03,  1.7763e-02,
            5.6305e-03, -3.4578e-02, -1.2965e-01,  1.3569e-03, -5.7878e-02,
            1.3272e-01, -4.1164e-02, -1.2947e-02,  5.1788e-02, -7.9227e-03,
            5.6931e-02, -7.7197e-03,  7.7456e-02,  1.8665e-02, -4.5540e-02,
            8.1336e-02, -3.0270e-02, -6.5412e-02,  4.3442e-02,  3.6134e-02,
           -4.1570e-02, -2.1811e-02,  1.1747e-01,  1.6132e-01, -6.0810e-02,
           -3.7713e-02, -7.5111e-02,  1.0810e-02, -7.7321e-02, -4.9938e-02,
            1.7424e-02,  4.4457e-02, -4.6775e-03,  1.9364e-02,  2.4180e-02,
           -1.7030e-02,  2.4586e-02,  3.5345e-02,  2.7631e-03,  2.5894e-02,
            1.5789e-01,  3.3225e-02,  6.0844e-03,  9.1937e-02, -3.8965e-03,
           -1.2146e-02, -7.7637e-02,  2.6616e-02, -9.6358e-02, -8.8464e-02,
            1.3770e-02,  4.2495e-02,  6.4239e-02, -1.0517e-02, -3.9067e-02,
            1.1808e-02, -1.1061e-01, -5.4720e-02, -6.4600e-02, -3.0157e-02,
           -5.4923e-03, -2.0830e-02, -5.1517e-02,  7.4073e-02,  1.0159e-01,
            2.6526e-02, -4.6871e-02, -3.0946e-02, -6.1070e-03,  4.3172e-02,
           -8.9039e-03, -6.3156e-02,  1.2203e-02, -1.1977e-02,  1.2595e-01,
            3.8074e-02,  5.5668e-02, -5.4720e-02, -3.3834e-02, -4.1480e-02,
           -1.7305e-01,  3.9991e-02,  1.3082e-01, -1.3146e-03,  1.7142e-02,
           -1.1855e-01, -2.8307e-02, -3.5841e-02,  3.7104e-02,  3.3495e-02,
            2.4563e-02, -2.6819e-02, -2.9029e-02, -6.5953e-02,  4.8360e-02,
           -1.4219e-01,  6.1758e-02,  5.4675e-02, -1.3479e-01, -8.2058e-02,
            7.1276e-03,  2.8037e-02, -4.9126e-02, -7.7817e-02, -2.4360e-02,
           -7.8810e-02,  1.4133e-03, -2.9368e-02,  2.0661e-02,  4.1818e-02,
            8.6524e-02,  3.1389e-06,  7.0464e-02,  6.1521e-03,  2.7473e-02,
           -4.8424e-04,  2.0300e-02,  1.1765e-01,  2.6571e-02,  5.4089e-02,
            9.8230e-03,  1.8090e-02, -5.8735e-02,  1.9556e-02,  7.3306e-02,
           -7.4479e-02,  2.5248e-03,  9.7351e-02, -2.3063e-02, -4.8314e-02,
            1.3691e-02, -7.7817e-02, -1.3777e-01, -3.6563e-02,  5.6886e-02,
           -3.7059e-02,  2.5736e-02, -4.4435e-02, -1.4553e-01, -1.9172e-02,
           -2.8826e-02, -4.8450e-02, -5.5532e-02,  4.9668e-02,  1.1485e-01,
            1.4041e-02, -3.6991e-02, -7.4434e-02,  4.1660e-02, -3.4933e-03,
           -2.9728e-02, -2.2161e-02,  6.7667e-03, -2.6480e-02, -1.0177e-01,
            6.4600e-02,  9.9877e-02,  6.2254e-02,  8.5847e-02, -8.1336e-02,
            6.3246e-02,  1.2649e-01, -1.2158e-02, -6.1983e-02,  8.3186e-02,
            1.8214e-03,  1.1693e-01, -6.7938e-02,  1.2413e-03,  6.7306e-02,
            4.7006e-02,  1.9984e-02, -9.2366e-03,  2.0199e-02,  2.5849e-02,
           -4.5833e-02, -2.2691e-02, -1.7842e-02, -6.0855e-02, -8.1291e-02,
            5.4720e-02, -9.1035e-02,  8.8982e-03,  4.1593e-02,  1.5755e-02,
           -3.1758e-02,  1.9804e-02,  4.6555e-02,  2.6458e-02, -1.1238e-02,
           -8.3140e-02,  1.8947e-02,  9.5997e-02,  6.5581e-03,  4.0803e-02,
            1.8699e-02,  6.3472e-02, -1.8349e-02, -1.7233e-02, -7.8178e-02,
            1.8202e-02,  2.9977e-02, -4.1999e-02, -5.7472e-02, -1.1188e-01,
           -8.6975e-02, -1.1408e-02,  9.4103e-02,  8.8419e-02,  3.7691e-02,
            7.1502e-02,  1.4199e-02,  9.1982e-02,  1.2415e-01,  3.2435e-02,
           -5.7607e-02, -3.8931e-02, -1.0944e-01, -8.6614e-02,  1.2649e-01,
           -3.6788e-02,  5.9502e-02,  6.7577e-02,  6.5231e-02, -3.3540e-02,
           -7.5068e-04], dtype=torch.float64),
      tensor([1, 0], dtype=torch.int8))
```

✓ Sample dataset using NullLoader

✓ Define Nulloader Algorithm

```
from functools import reduce
import torch
from torch.utils.data import DataLoader
```

```

class NullLoader(DataLoader):
    def __init__(self, proto_loader:DataLoader,
                  outbatch:int, rejection_iters:int, buffer_len:int,
                  reduced_shape:tuple[int], outshape:tuple[int],
                  dimreduction = lambda x: x,
                  device = 'cuda'):
        self.proto = proto_loader
        self.proto_batch = proto_loader.batch_size
        self.mem = buffer_len
        self.dimreduce = dimreduction
        self.rdim = reduced_shape
        self.odim = outshape
        self.batch_size = outbatch
        self.rejection_iters = rejection_iters
        self.device = device
        # assert outbatch % rejection_iters == 0, f"must be able to make full batch in {rejection_iters} iters"
        # assert self.mem % self.proto_batch == 0, f"buffer len {self.mem} must be divisible by the prototype loader
        self.buffer = torch.zeros((self.mem,) + self.rdim, device=self.device)

    def __iter__(self):
        cand_labels = []
        candidates = []
        gradweights = []
        top_n = self.batch_size // self.rejection_iters

        for _ in range(self.rejection_iters):
            # print("rejection iter", _)
            self._protobatch, self._protolabels = next(iter(self.proto))
            self._protobatch, self._protolabels = self._protobatch.to(self.device), self._protolabels.to(self.device)
            dimreduced = self.dimreduce(self._protobatch)

            # Compute the SVD of the buffer
            U, S, Vh = torch.linalg.svd(self.buffer.view((self.mem, -1)), full_matrices=False)

            # Determine the rank and construct the projection matrix onto the null space
            threshold = 1e-6
            rank = (S > threshold).sum().item()
            if rank == 0:
                # The buffer is empty or has rank zero; the null space is the entire space
                P_null = torch.eye(self.buffer.shape[1], device=self.device)
            else:
                V_row = Vh[:rank, :]
                P_row = V_row.T @ V_row
                P_null = torch.eye(P_row.shape[0], device=self.device) - P_row

            # Project candidates onto the null space and compute projection errors
            dimreduced_flat = dimreduced.view(self._protobatch.size(0), -1).T # Shape: (n_features, batch_size)
            projections = P_null.double() @ dimreduced_flat # Shape: (n_features, batch_size)
            proj_errs = torch.linalg.norm(projections, dim=0)

            # Select top_n candidates with the largest projection errors
            asort = torch.argsort(proj_errs, descending=True).to(self.device)
            gradweights.append(proj_errs[asort])
            candidates.append(self._protobatch[asort])
            cand_labels.append(self._protolabels[asort])

            # Update the buffer
            self.buffer = torch.roll(self.buffer, shifts=top_n, dims=0)
            self.buffer[:top_n] = dimreduced[asort[:top_n]].float()

        # Concatenate and normalize gradweights
        candidates = torch.cat(candidates, dim=0)
        cand_labels = torch.cat(cand_labels, dim=0)
        gradweights = torch.cat(gradweights, dim=0)
        gradweights /= gradweights.sum()

        yield candidates, cand_labels, gradweights, asort

```

✓ Sample dataset using Nulloader

```
# Create a DataLoader to load the dataset in batches
BATCH = 32
# print("dataset length", len(private_dataset))
# control_loader = torch.utils.data.DataLoader(private_dataset, batch_size=BATCH, shuffle=True)
# print("dataset length", len(ellipse_dataset))
# control_loader = torch.utils.data.DataLoader(ellipse_dataset, batch_size=BATCH, shuffle=True)
print("dataset length", len(synthetic_dataset))
control_loader = torch.utils.data.DataLoader(synthetic_dataset, batch_size=BATCH, shuffle=True)
def norm(x):
    vnorm = torch.linalg.vector_norm(x.view(x.shape[0], -1), dim=1)
    return x.transpose(0, -1).div(vnorm).transpose(0, -1)
PROTOBATCH = 16
RITERS = 8
MEM = 256
proto_loader = torch.utils.data.DataLoader(private_dataset, batch_size=PROTOBATCH, shuffle=True)
print("loader", proto_loader.batch_size)
exp_loader = Nulloader(control_loader, BATCH, RITERS, MEM, (256,), (256,), norm, device="cuda")
# print(exp_loader.get_scores())
for i in range(1):
    print("iter", i)
    candidates, labels, weights, asort = next(iter(exp_loader))
# for i in exp_loader:
#     pass
print(candidates.shape)
print(weights.shape)
print(asort.shape)

# get top 100 essays
selected_labels = labels[:100]
```

```
dataset length 296
loader 16
iter 0
torch.Size([256, 256])
torch.Size([256])
torch.Size([32])
```

> Perform K-Means Clustering

```
[ ] ↪ 1 cell hidden
```

✓ Analysis

✓ Randomly Select Labels and actual distribution as a control

```
num_samples = 100 # Set number of samples
dataset = synthetic_dataset
dataset_length = len(dataset)

# Now randomly sample as a control
sampled_indices = random.sample(range(dataset_length), num_samples)

# Get the sampled data points
randomly_selected = [dataset[i] for i in sampled_indices]

randomly_selected_labels = []
for features, label in randomly_selected:
    print(label)
    randomly_selected_labels.append(label)
```

```

actual_labels = []
for i in range(len(dataset)):
    _, labels = dataset[i]
    actual_labels.append(labels)

↩ tensor([4, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([0, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([0, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([0, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([0, 0], dtype=torch.int8)
  tensor([1, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([0, 1], dtype=torch.int8)
  tensor([4, 0], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([3, 1], dtype=torch.int8)
  tensor([4, 1], dtype=torch.int8)
  tensor([1, 1], dtype=torch.int8)
  tensor([3, 0], dtype=torch.int8)

```

✓ Analyze results

```

list_to_analyze = selected_labels

races = ["Hispanic", "Black", "American Indian or Alaskan Native", "Asian", "White"]
genders = ["Female", "Male"]

race_dict = {}
gender_dict = {}

```

```

for label in list_to_analyze:
    race_dict[races[label[0]]] = race_dict.get(races[label[0]], 0) + 1
    gender_dict[genders[label[1]]] = gender_dict.get(genders[label[1]], 0) + 1

# print percentages
print("Selected Distribution")
print("Race distribution:")
for race in races:
    percentage = (race_dict.get(race,0) / len(list_to_analyze)) * 100
    print(f"{race}: {percentage:.2f}%")

print("Gender distribution:")
for gender in genders:
    percentage = (gender_dict.get(gender, 0) / len(list_to_analyze)) * 100
    print(f"{gender}: {percentage:.2f}%")

random_race_dict = {}
random_gender_dict = {}

for label in randomly_selected_labels:
    random_race_dict[races[label[0]]] = random_race_dict.get(races[label[0]], 0) + 1
    random_gender_dict[genders[label[1]]] = random_gender_dict.get(genders[label[1]], 0) + 1
print()
print("Random Distribution")
print("Race distribution:")
for race in races:
    percentage = (random_race_dict.get(race, 0) / len(randomly_selected_labels)) * 100
    print(f"{race}: {percentage:.2f}%")


print("Gender distribution:")
for gender in genders:
    percentage = (random_gender_dict.get(gender, 0) / len(randomly_selected_labels)) * 100
    print(f"{gender}: {percentage:.2f}%")

actual_race_dict = {}
actual_gender_dict = {}

for label in actual_labels:
    actual_race_dict[races[label[0]]] = actual_race_dict.get(races[label[0]], 0) + 1
    actual_gender_dict[genders[label[1]]] = actual_gender_dict.get(genders[label[1]], 0) + 1
print()
print("Actual Distribution")
print("Race distribution:")
for race in races:
    percentage = (actual_race_dict.get(race,0) / len(actual_labels)) * 100
    print(f"{race}: {percentage:.2f}%")

print("Gender distribution:")
for gender in genders:
    percentage = (actual_gender_dict.get(gender, 0) / len(actual_labels)) * 100
    print(f"{gender}: {percentage:.2f}%")

```

 Selected Distribution
 Race distribution:
 Hispanic: 8.00%
 Black: 9.00%
 American Indian or Alaskan Native: 2.00%
 Asian: 49.00%
 White: 32.00%
 Gender distribution:
 Female: 53.00%
 Male: 47.00%

Random Distribution
 Race distribution:
 Hispanic: 11.00%
 Black: 4.00%
 American Indian or Alaskan Native: 1.00%


```
Asian: 44.00%  
White: 40.00%  
Gender distribution:  
Female: 47.00%  
Male: 53.00%
```

```
Actual Distribution  
Race distribution:  
Hispanic: 10.14%  
Black: 4.73%  
American Indian or Alaskan Native: 0.68%  
Asian: 47.30%  
White: 37.16%  
Gender distribution:  
Female: 50.00%  
Male: 50.00%
```

Final writeup

As you can see the results are decent. The percentage of black applicants doubled from the actual distribution, the number of white applicants decreased, and the american indian and alaskan native population despite being wildly underrepresented grew by almost 4x. Unfortunately, the number of hispanics went down from the original distribution showing the algorithm isn't perfect but still has potential.